Parallel I/O Ports

Parallel I/O ports can be seen as extensions of the CPU's data bus. They transfer one word at a time between the CPU and a peripheral. A parallel interface usually involves additional "handshaking" lines and a well-defined protocol to control the transfer of data. Parallel interfaces are used to transfer data with higher-speed peripherals such as printers. We will study one simple example of a parallel interface, the parallel printer interface.
After this lecture you should be able to: (1) design simple input, output and bidirectional I/O ports using registers, tri-state buffers and and open-collector buffers; (2) describe the operation of a parallel printer interface; and (3) write C programs to read and write the individual bits of an I/O port.

I/O Ports

All microcomputer-based control systems must have input/output (I/O) devices to move data between the outside world and the computer. The interface between the CPU and these I/O devices is through I/O ports that appear as memory locations to the CPU. Using these I/O ports the CPU can input (read) or output (write) a number of bits (typically one byte) at a time.

Typical examples of I/O ports include output ports that drive LEDs, ports to scan a keypad, ports to control machinery, etc. More complex I/O interfaces such as floppy disk controllers or serial interface chips usually contain several I/O ports. Some ports are used to obtain status information about the interface through "status registers" and other ports can control the interface's operation through "control registers."

For example, the printer interface on the IBM PC has associated with it a status port that can be used to obtain certain status information (busy, on-line, out of paper, etc). The printer interface also has a control port that can be used to reset the printer and set the automatic line feed mode. In addition, there is an output port that is used to output the character to be printed.

Software Aspects

High-level languages such as C don't allow the programmer to read or write specific memory locations. Special functions (often called `peek()` and `poke()`) are used to access the memory locations corresponding to the I/O ports.

It's often necessary to set or clear a particular bit on an output port or to test the value of a particular bit on an input port. This can be done with bit masks and the bit-wise logical operations AND and OR.

To set a particular bit(s), the current output value is OR'ed with a bit-map which contains 1's in the bit positions to be set. To clear a particular bit(s), the current output value is AND'ed with a bit-map which contains 0's in the bit positions to be cleared. To test the value of a particular bit, the input value is ANDed with a bit-map which contains 1's in the bit position(s) to be tested.

Here are some examples of C code that access I/O ports and manipulate the bits:

```
unsigned char c ;
...
c = peek ( 0x60 ) ; /* read byte from address 0x60 */
...
if ( c & 0x80 ) {   /* test MS bit */
...
if ( c & 0x07 ) {   /* check LS 3  bits */
...
c = c | 0x7 ;       /* set LS 3 bits to 1s */
...
c = c & 0xcfh ;     /* clear bits 5 and 4 */
...
poke ( 0x70, c ) ;  /* write to I/O port at 0x70 */
```

Often it's not possible to read the value written to an output port (i.e. the port is write-only). If individual bits will need to be changed, it's necessary to keep track of the most recent value written to the port, modify this copy and then write the result to the I/O port location.

For example, the following code clears the LS bit of a value that is being output to an 8-bit output port which is located at address 0x80. In this case the port is output-only so a copy of the output value is kept in the variable `outval`.

```
outval = outval & 0x0fe ;
spoke ( 0x80, outval ) ;
```

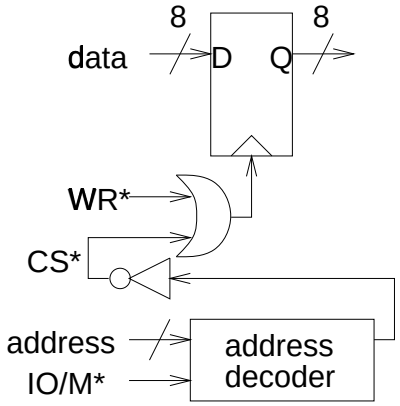
Exercise: The status port for a serial interface chip is located at I/O port 0x55. Bit 2 (bits are usually numbered from 0 starting with the LS bit) will have the value 1 if a received character is available to be read (from another port on the chip). Write a section of C code that checks to see if there is a character ready to be read.

Implementation of I/O Ports

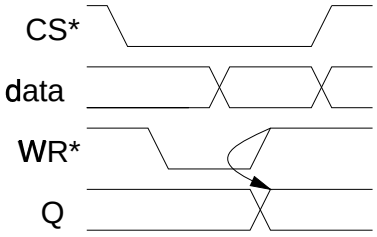
Output

Output ports are implemented using registers – multi-bit flip-flops with a common clock. The register's data inputs (D) are connected to the CPU data bus and the register's clock input is driven by the CPU write strobe (WR*). In addition, an address decoder is used to make sure the clock is only asserted when the CPU is addressing the desired IO or memory address. The rising edge of the write strobe loads the data into the register output (Q) and this output stays fixed until the register is written again.

The following schematic shows how a register could be connected to operate as an output port. The CPU's write strobe (WR*) is used to clock the data into the register, but only if the address on the CPU bus corresponds to the address of the output port:



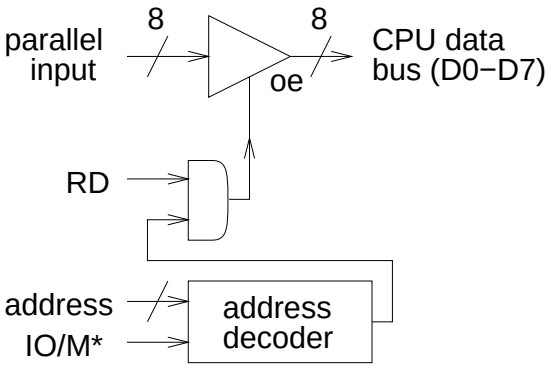
The following timing diagram shows the relationship between the signals. Note that the output is held after the rising edge of the write strobe (WR*):



Input

Input ports can also be implemented with a minimum of hardware. A tri-state buffer is used to connect the external digital input to the CPU's data bus during a read cycle if the CPU is addressing the memory or IO address assigned to the input port. The read strobe (RD*) is used to enable the buffer so that it connects the input to the CPU data bus.

The following schematic shows how a register could be connected to operate as a parallel input port. The CPU's inverted RD* strobe (RD) is used to enable the output of a tri-state[1] buffer when RD is active and the address corresponds to the address of the input port:



The value read by the CPU will be the value on the input port at the time that the peek() function was called.

Bi-Directional I/O Ports

By using open-collector[2] a outputs on an output port it's possible to use the same signal pins for both input and output. The open collector outputs are driven high by pull-up resistors and can be driven low by either the output port or by an external device. An

[1] A tri-state output can be in the normal high and low states as well as a third, "high-impedance," state. In this state the output is effectively disconnected from the rest of the circuit.

[2] An open-collector output can only pull its output to ground, it cannot drive it to a high level.

input port is attached to these lines. The state of the I/O interface lines can be read by reading the input port.

Exercise: To what value must the outputs be set in order to be able to read from an external device?

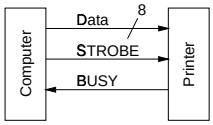
Address and I/O Decoding

The design of address decoders for I/O ports is similar to the design for memory systems. A typical I/O interface will only require a few (typically less than 16) ports (addresses). On some CPUs (such as those from Intel) there are separate I/O and memory address spaces. In this case the decoder must enable the port only for the appropriate address space.

"Centronics" Parallel Printer Port

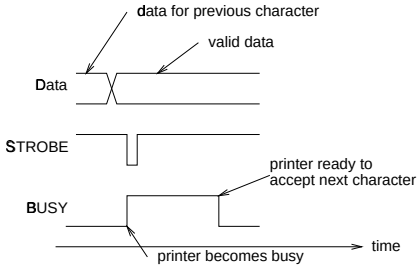
We will use the "Centronics"-compatible parallel interface as an example of a parallel interface. Other parallel interfaces such as SCSI and IEEE-488 operate in a similar fashion but have more complicated protocols to allow the interface to be shared by several peripherals.

The parallel printer interface was designed to allow computers to drive printers. There are eight data lines, four output (to printer) handshaking signals and five input (from printer) handshaking signals. Of the handshaking signals, only one input signal (BUSY) and one output signal (STROBE*) are required. The other handshaking signals are used for things such as out-of-paper, on-line, and error signals.



To write a value to the printer the data bits are put on the eight data lines (D0 to D7) and the STROBE* output signal is set low for a minimum of 0.5 μseconds. When the STROBE* goes low, the data is accepted by the printer and the printer turns the BUSY line high to indicate that it has accepted the character, that the printer is busy and that no more data should be sent.

When the printer has finished processing the character it turns the BUSY line back to a low level and the computer can then send the next character.



This interface uses TTL signal levels (about 0 volts for low and about 5 volts for high).

There are additional handshaking lines to control various printer features (e.g. auto line feed) and to indicate various printer status conditions (e.g. out of paper).

There original IBM PCs parallel port was an output-only Centronics-compatible interface but in recent designs the port can also be configured as an input. The maximum speed usually depends on the software use but is typically 50 to 100 kB/s.

Small Computer System Interface (SCSI)

This is a type of parallel interface that allows for bidirectional data transfer and for up to 8 hosts ("initiators") and peripherals ("targets") to be connected together on the same bus. The SCSI interface is well defined and and is available on many different computers. It is widely used to connect computers to disk and tape drives, CD-ROMs, scanners, high-speed printers, etc.

The SCSI interface includes a protocol for arbitrating access to the bus by initiators and for selecting a specific target. The actual data transfers over the SCSI bus use a similar request/acknowledge protocol with each byte transfer being acknowledged by the target before another byte is transferred.

Depending on the speed of the peripheral and the host interface the bus can transfer data at up to several megabytes per second. The SCSI devices attached to the bus are electrically connected up in parallel with each device configured to respond to a particular bus ID number (ID).

The physical interface uses a 50-pin connectors with two connectors on each device so that they can be daisy-chained. Because of the high bus speed care has to be taken to properly terminate the bus in it's characteristic impedance to minimize reflections.

Like the Centronics interface the SCSI bus also uses TTL-level signals but it needs open-collector or tri-state drivers.

Another advantage of the SCSI interface is that it defines a set of common commands for devices with similar characteristics. This allows the same software to drive different devices. For example, the same generic commands (rewind, skip forward, etc) can be used to control tape drives from different manufacturers.

Although a SCSI interface can be built using a simple parallel interface and programmed i/o, this type of interface is too slow for most applications. SCSI interfaces are available that implement the SCSI protocol and can transfer data using directly from the SCSI bus to memory using (direct memory access - DMA).

HPIB/GPIB/IEEE-488

The General Purpose Interface Bus (GPIB) is another bidirectional interface. Like the SCSI bus it allows multiple bus masters ("talkers") and slaves ("listeners"). It was developed by HP who named it HPIB (HP Interface Bus). The standard is called IEEE-488. This bus is used mostly for control of laboratory instruments.